QA Consulting

Introduction to Git.

TRAINING MATERIALS - COURSE HANDOUT

Contacts

devdatta.gonsai@qa.com

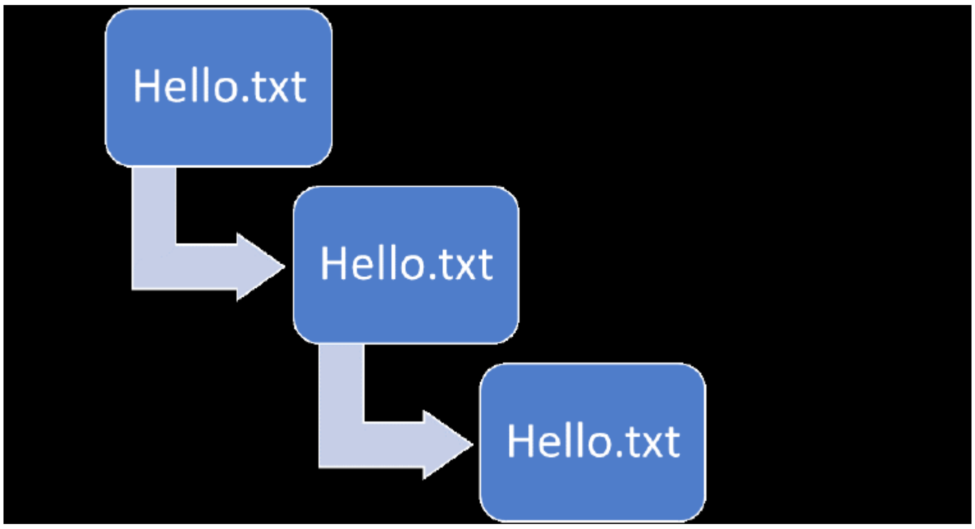
team.qac.all.trainers@qa.com

www.consulting.qa.com

Contents

What is a Version Control System?

A Version Control System (or VCS) is a tool that records the history of changes to a file. This repository of versioned code and files can then have their changes reviewed and compared, as well as being able to revert to previous versions. For example:



These VCS tools are sometimes referred to as Source Code Management (SCM) tools, as their main use within software development is to track changes of code. These tools typically support any and all types of code and file.

There are 2 main types of version control systems used within the industry today, which are:

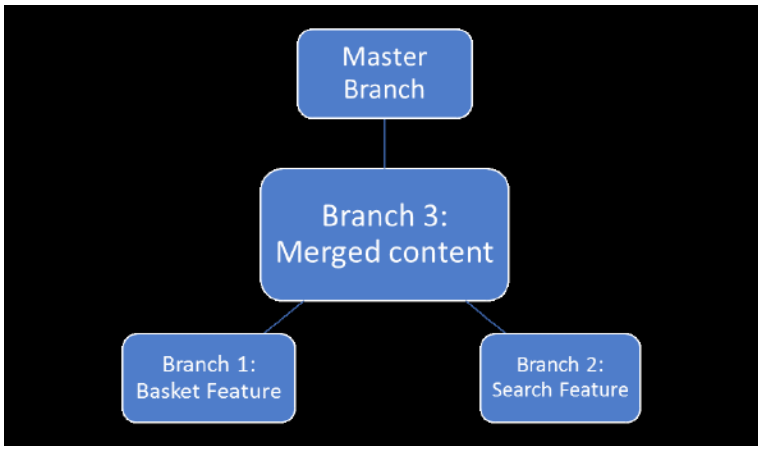
- Centralised Version Control Systems (CVCS)
- Distributed Version Control Systems (DVCS)

Admittedly, DVCS' are used more prominently within business today, but both systems have their uses (see section x)

Benefits of a VCS

There are many benefits of using a VCS within a project. Some of these benefits include:

- **Change history** – the changes of a file are completely transparent within a VCS, and easily identifiable. As well as this, metadata (information about the files or data about the data) is maintained, meaning that we can see when a file was changed and uploaded (or pushed). With this type of data, we can see what we wish to revert, compare or change within the files.
- **Branches** – we can separate a project within a VCS through the use of branches. A branch on a VCS is simply an independent stream of work, which can then be combined (or merged) with other branches.



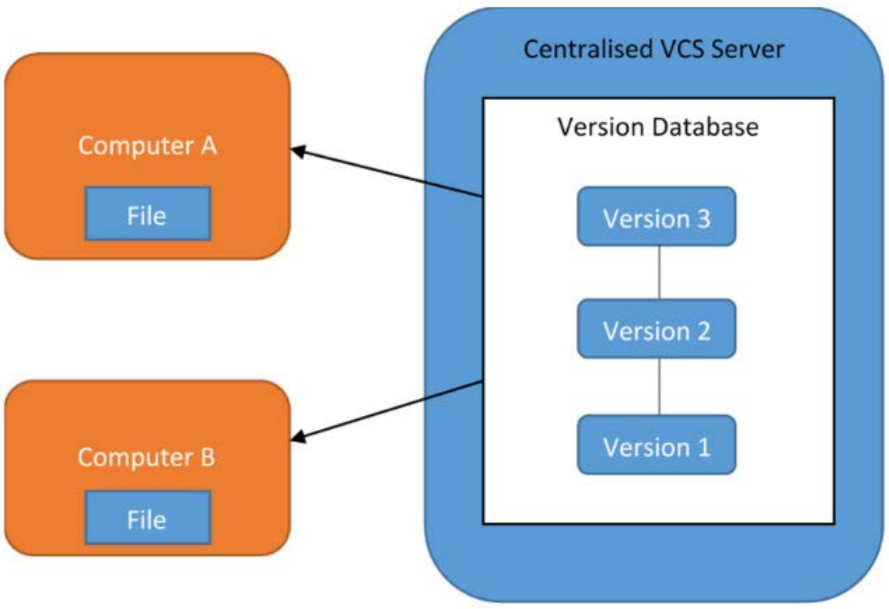
Let's assume we are creating an e-Commerce website for a client. Within this website, there are a number of components, or features, that will need to be included for the product to function correctly.

Therefore, we can dedicate a branch to each of these features, such as the shopping basket and the search function, and push our code to these features. These can then be merged together in another branch if needed.

- **Traceability** – we can easily identify the author of a file as well as the author of any pushes to a file. This allows for easier identification of where a problem could lie, and helps with root cause analysis.

Centralised Version Control Systems (CVCS)

A Centralised Version Control System (CVCS) is an architecture option for Version Control Systems. It uses a central server as a hub to work from, with anyone who wishes to use the VCS needing to connect to it. The server contains a database with all versioned content, which the different computers can access for the versioned files.

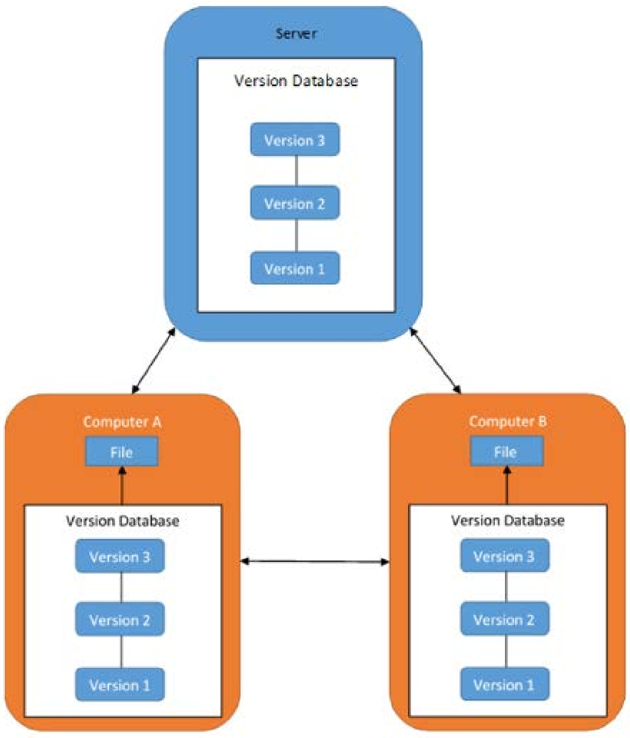


The main benefit of this type of set up is the administration around files. Due to the files only being on one central server before being checked out, there can be a good layer of security on the server.

The main downside of a CVCS is the Single Point of Failure – or a SPOF. If the central server containing your versioned files goes down or brakes for any reason, files are lost and backups are assumed to be lost as well.

Distributed Version Control Systems (DVCS)

A Distributed Version Control System (DVCS) is another architecture option for Version Control Systems. Whilst all code is held on a central server, the full repository is copied, or mirrored, onto any computer that is working on the SCM.



When the repository is mirrored, we are essentially creating a full back up of the VCS. This alleviates the issue of Single Point of Failure, as we are creating back up's of the repository as soon as it is mirrored. If the server was to go down for one reason or another, the back-up's on the local machines can be used to restore the server.

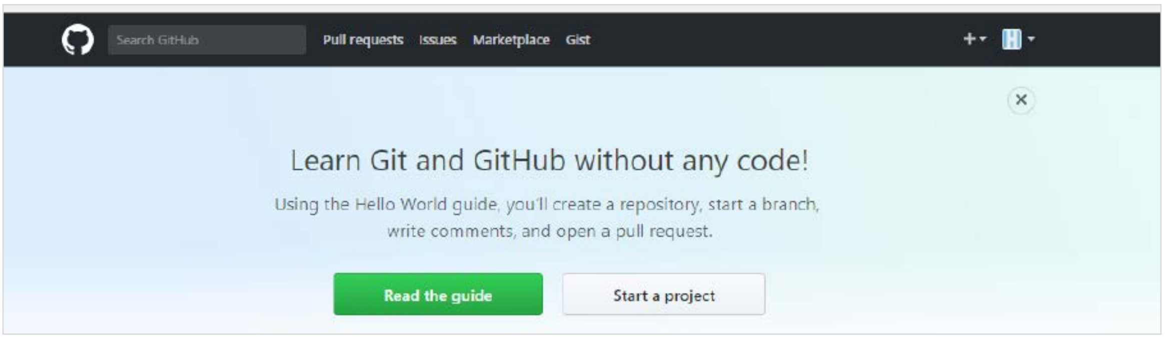
Git

INTRODUCTION

Git is one of the most popular distributed version control systems in the IT industry. It's used by thousands of software developers, both on a hobby and enterprise level. Git can be used as a stand-alone tool, or it can be integrated within supported IDE's, such as Eclipse.

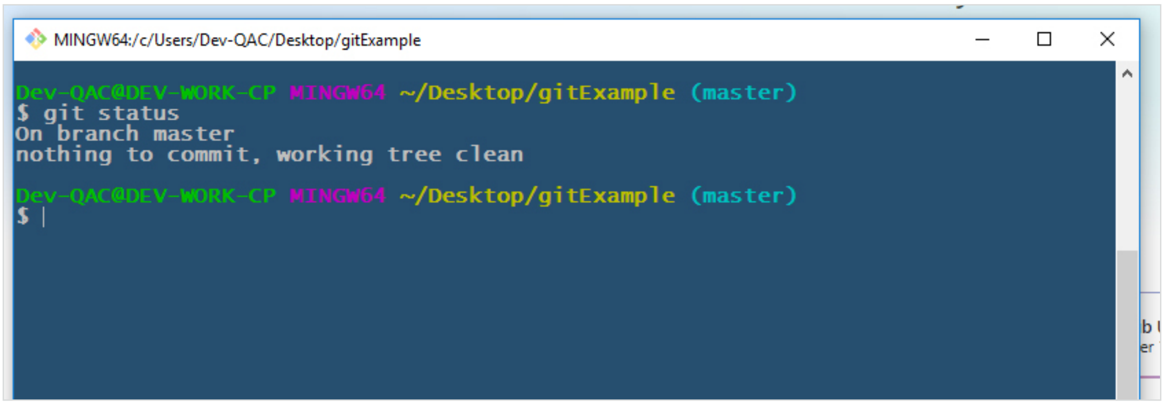
Linus Torvalds, the developer of the original Linux kernel, was the developer of Git. The initial idea of Git was to support large Linux kernel project, with the mindset of VCS in tune. However, as the tool became more popular, Torvalds further developed the tool to support a multitude of project types.

An important idea to remember about Git is that it is used at both an open-source and enterprise level. There are many independent users of Git, as well as many open-source projects. On the other side of the spectrum, there are many large corporations that use Git to support software development.



Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.



```
Dev-QAC@DEV-WORK-CP MINGW64 ~/Desktop/gitExample (master)
$ git status
On branch master
nothing to commit, working tree clean

Dev-QAC@DEV-WORK-CP MINGW64 ~/Desktop/gitExample (master)
$ |
```

Git is supported through GitHub, which is a web-based UI of Git. It provides a number of controls on the website that will help with the use of Git, from starting a new repository to maintaining several branches on a long running project. As well as this, there is a command line interface, or CLI, used for Git to allow for quicker actions around pushing and pulling code, and managing your repository.

BENEFITS OF GIT

Performance – Git allows for many quick actions revolving around your repository. This is done through the intuitive web-based GUI, as well as the speed of writing commands through the CLI.

Offline work – As soon as you mirror your repository from Git, you can work offline without the need to have a constant connection with the central server. This allows for greater flexibility of work, as well as the ability to work without the threat of a connection error negatively altering your work.

Flexibility – The flexibility of Git is in reference to the support around small and large projects. It's very easy to work with small applications through Git, but as your project expands, Git will scale alongside it, allowing for individual workflow management through branches.

GitHub

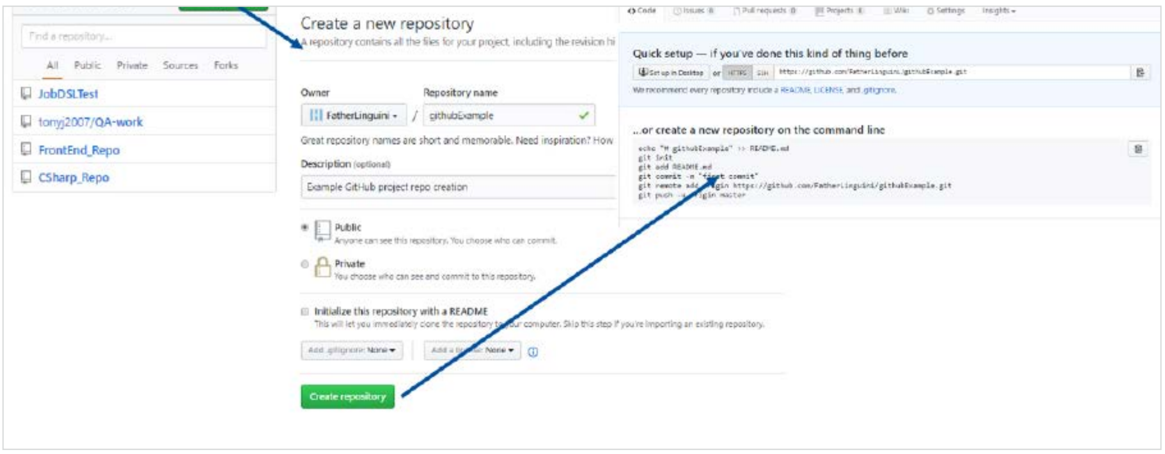
INTRODUCTION

GitHub is a web based GUI to interact with Git.

You can create repositories through GitHub, as well as add, edit and delete files in your repositories. You can also create branches for individual workflows within your repositories.

There are many open source projects that are hosted on GitHub. From here, you can view source code of an application or copy a repository – known as forking – to use this as a base for your own project.

CREATING A NEW REPOSITORY



Create a new repository
A repository contains all the files for your project, including the revision hi

Owner: FatherLinguini / Repository name: githubExample

Great repository names are short and memorable. Need inspiration? How

Description (optional): Example GitHub project repo creation

Public
Anyone can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None

Quick setup — if you've done this kind of thing before
Set up in Desktop or HTTPS SSH https://github.com/FatherLinguini/githubExample.git
We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# githubExample" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/FatherLinguini/githubExample.git
git push -u origin master
```

- **1.** On GitHub, click the button “New Repository”. You will be brought to a screen with a number of options.
- **2.** From here, you can fill out the information about your repository
  - › **a.** Owner/Repo Name – who is the admin of the repo and what the repo is called
  - › **b.** Description – a short introduction of your repository’s role and use
  - › **c.** Public/Private – if public, this repo is open to all. If private, access to this repo is restricted. (NOTE: private repositories are available to GitHub enterprise users only).
  - › **d.** Initialise with README – you can clone the repository immediately rather than having to initialise the repo and do an initial commit/push
  - › **e.** Add .gitignore – you can add a folder that will hold files that are ignored when you update the repo
  - › **f.** Add a license – this allows you to add more information around the use of your repository in a more official manner than something like a README.
- **3.** Once this is done, click create repository and you’ll get a number of steps to follow to continue with the creation and use of the repo.

Bitbucket

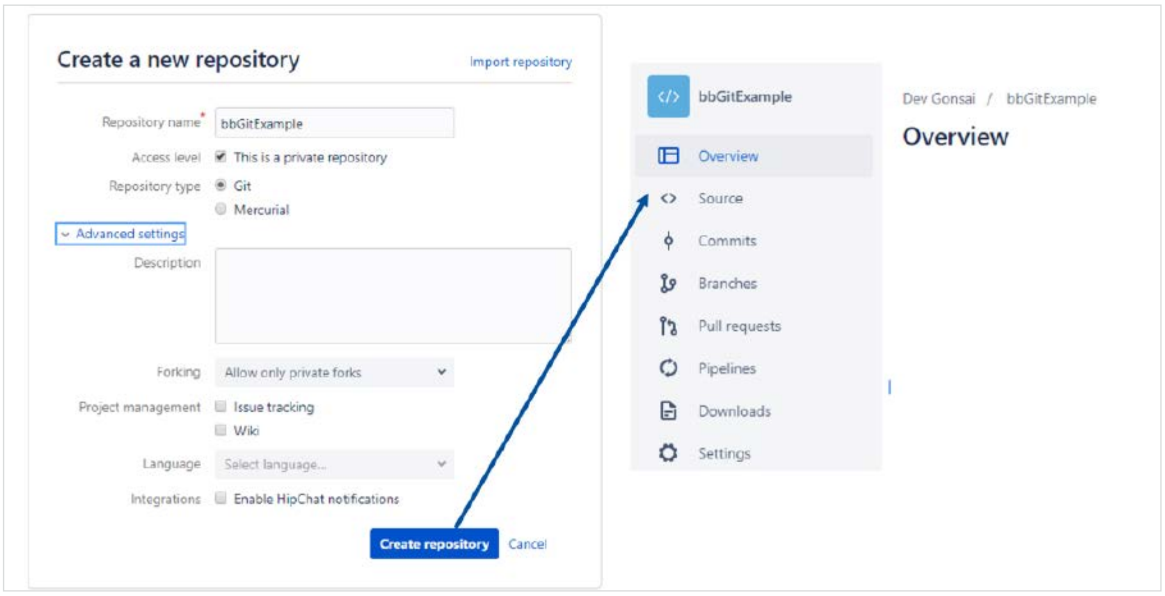
INTRODUCTION

Bitbucket is an alternative UI for Git based repositories.

It is maintained by Atlassian, one of the biggest companies in the Continuous Integration and DevOps tech space. Other tools they author and/or maintain include Jira, Confluence, Bamboo and HipChat.

Bitbucket is used by many businesses in the IT industry. They offer most, if not all, features that GitHub do, but have the added bonus of the integration with the rest of the Atlassian tools aforementioned.

CREATING A NEW REPOSITORY



Create a new repository

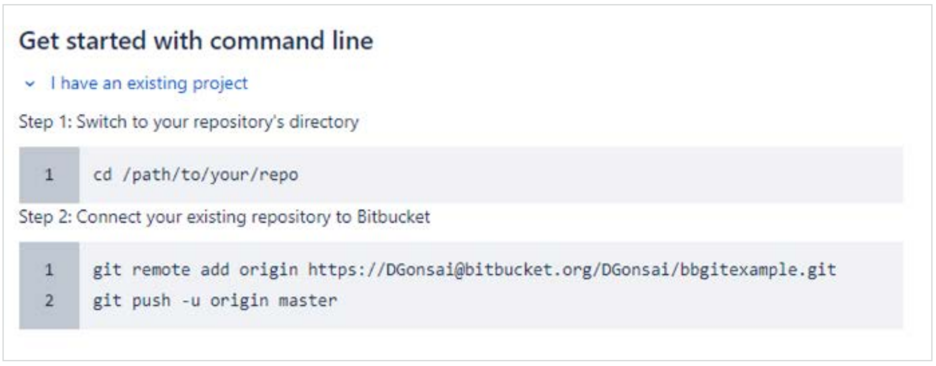
Repository name*: bbGitExample
Access level: ☑ This is a private repository
Repository type: ● Git ○ Mercurial
Advanced settings
Description:
Forking: Allow only private forks
Project management: ☐ Issue tracking ☐ Wiki
Language: Select language...
Integrations: ☐ Enable HipChat notifications

- **1.** Creating a new repository is very similar to GitHub, in that the same information is generally required.
  - **a.** Repo Name – name of the repository that will be used
  - **b.** Access level – you can make this public or private with a check box
  - **c.** Repository type – Bitbucket supports both Git and Mercurial based repositories within its site.
- **2.** There are some advanced settings you may want to look at:
  - **a.** Description – short piece of information about your repo
  - **b.** Forking – you can allow for forks (or copies) of your repo to be public, private, or neither
  - **c.** Project Management – this is where you can integrate a Project Management tool within your repo, such as Jira.
  - **d.** Language – base language of you project. Note: this is in relation to the programming language, not the communication language!
  - **e.** Integrations – this is way to integrate another tool within the repo. In this case, it is HipChat, which is an instant messaging service. This would enable notifications in the chat in relation to particular files being changed in some way.

Bitbucket - Getting Started

INTRODUCTION

BitBucket gives you a set of instructions to follow to initialise your repository. To initialise your repo means that you are doing the starting set up that is required before you can fully start using the repo. Some of the commands needed for this are listed here:



Get started with command line

I have an existing project

Step 1: Switch to your repository's directory

```
cd /path/to/your/repo
```

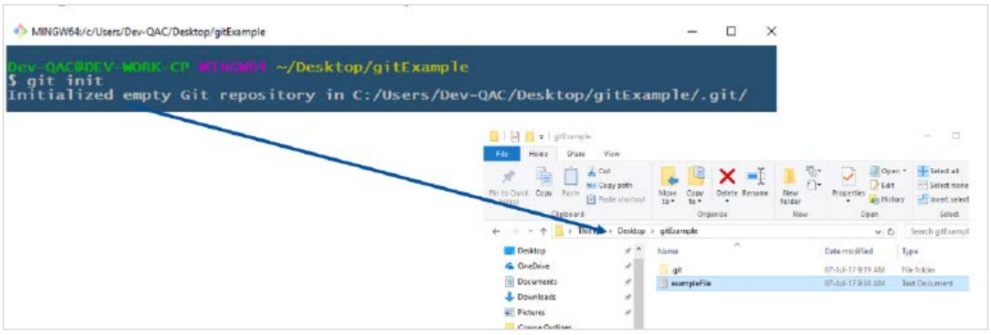
Step 2: Connect your existing repository to Bitbucket

```
git remote add origin https://DGonsai@bitbucket.org/DGonsai/bbgitexample.git
git push -u origin master
```

Git Commands

GIT INIT

You'll need to create a folder and use Git Bash to navigate to this folder (or just open up Git Bash within this folder). From here, enter git init, which begins the initialisation of the folder to be used with Git.



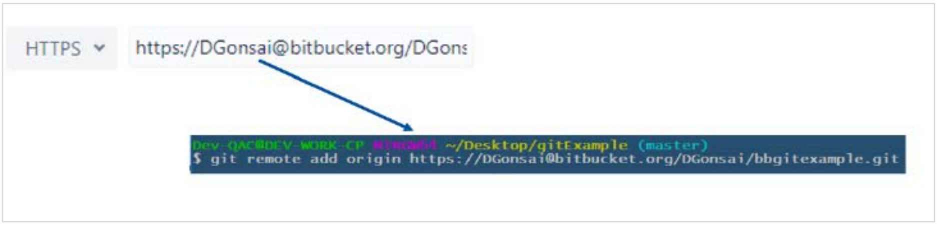
MINGW64:/c/Users/Dev-QAC/Desktop/gitExample

```
Dev-QAC@DEV-WORK-CP MINGW64 ~/Desktop/gitExample
$ git init
Initialized empty Git repository in C:/Users/Dev-QAC/Desktop/gitExample/.git/
```

This will create a .git folder within your directory, which contains information about version control, remote repositories, commit history, etc. Do NOT edit the content of this folder.

GIT REMOTE

You will then need to point to where your repository is. Your repo is on Bitbucket, so you will need to copy the HTTPS URL from Bitbucket and use it with the Git remote command, as such:



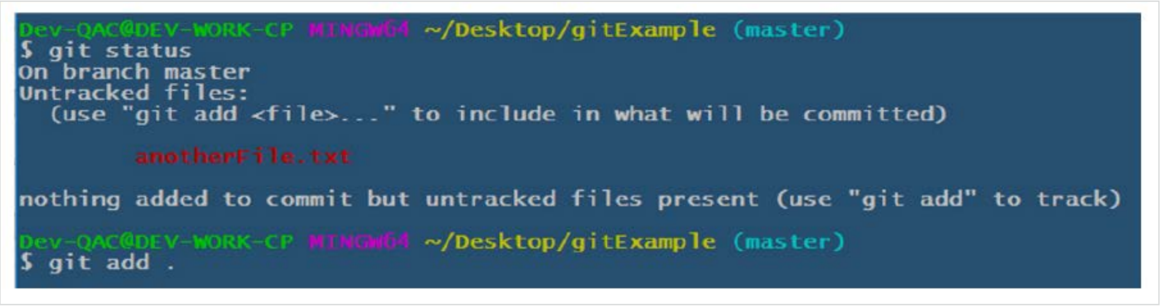
```
Dev-QAC@DEV-WORK-CP MINGW64 ~/Desktop/gitExample (master)
$ git remote add origin https://DGonsai@bitbucket.org/DGonsai/bbgitexample.git
```

Breaking down this command:

- **Git remote** – this is the command, which can take arguments such as add and remove
- **Add** – this is specifying that we are adding a new repository as an endpoint to push files to
- **Origin** – this is the reference of the endpoint – similar to a variable. Therefore, when we reference the origin keyword within a Git remote command, we are referencing the URL that follows
- **[URL]** – this is the URL of your repository on Bitbucket or Github, and will be stored in the keyword used for your endpoint (in this case, origin)

GIT ADD

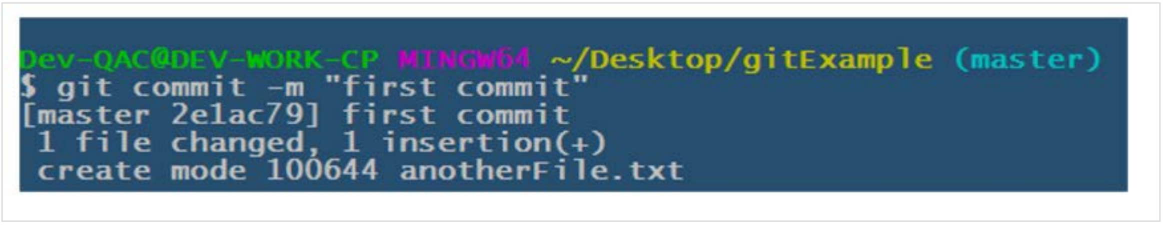
This adds files that are ready to be pushed to the repository. This can be individual files or all files. If you want to add all files, use the command git add . – the period specifies that you are adding everything within the initialised folder.



```
Dev-QAC@DEV-WORK-CP MINGW64 ~/Desktop/gitExample (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        anotherFile.txt

nothing added to commit but untracked files present (use "git add" to track)

Dev-QAC@DEV-WORK-CP MINGW64 ~/Desktop/gitExample (master)
$ git add .
```

GIT COMMIT

This stages the files that have been added through Git Add and indexes it in a new commit. These commits will be stored and the history of which files are staged and committed together can be viewed later.

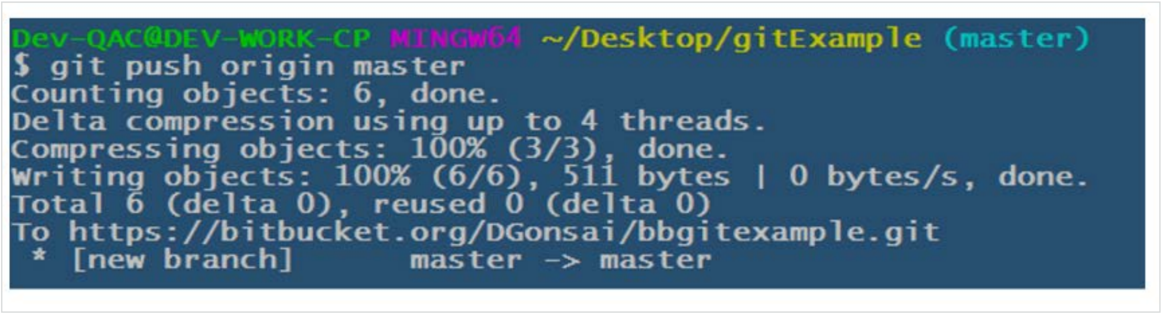


```
Dev-QAC@DEV-WORK-CP MINGW64 ~/Desktop/gitExample (master)
$ git commit -m "first commit"
[master 2e1ac79] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 anotherFile.txt
```

The “-m” portion of the command is specifying an accompanying message for the commit. You must add some form of message for a commit, as this will specify why this file was committed to the rest of the team working on it.

GIT PUSH

This will find all of the files in the most recent indexed commit and upload them, or push them, to the repository specified in your endpoint. Any local references will be the newest version of the remote references – i.e. any local files will be the newest version of the remote files on your Bitbucket site.



```
Dev-QAC@DEV-WORK-CP MINGW64 ~/Desktop/gitExample (master)
$ git push origin master
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 511 bytes | 0 bytes/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To https://bitbucket.org/DGonsai/bbgitexample.git
 * [new branch]      master -> master
```

Breaking down this command:

- **Git push** – this is the command, which can take additional arguments. These include force, mirror, and all
- **Origin** – this is the endpoint reference created in the remote command written earlier, which is the URL of your remote repo
- **Master** – this is specifying the branch to which you are pushing your files. This could be any branch in the future, but as this repo was just created, master is the only branch present.

BITBUCKET – AFTER THE PUSH

If we look on the Bitbucket repository, you'll notice that your file is now available to view! There are a number of options with this:

- **Overview** – general information around the repository, including when it was last access, who accessed it, the URL and an overview of pull requests, branches and forks.
- **Source** – you can view the source code and file contents uploaded through this option
- **Commits** – see the history of commits and pushes to this repo
- **Branches** – view all of the branches within your repo
- **Pull requests** – see who has made a pull request, which is essentially asking for permission to make changes to files before making the changes and pushing them
- **Pipelines** – see how this repo is connected to other tools, such as CI servers, project management tools and more
- **Settings** – general global settings of your repo

Git Hooks

INTRODUCTION

Git hooks are scripts that are executed by Git before or after key events such as: commit, push, and receive. Hooks are already built into Git, so do not need to be downloaded, and are run locally on your machine.

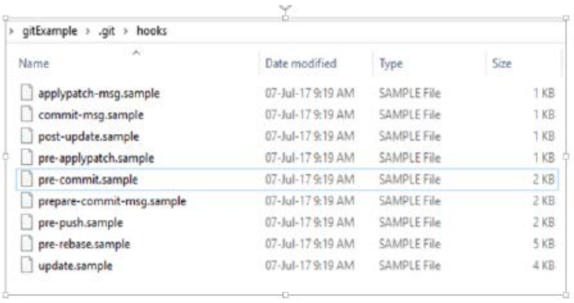
Hook scripts are completely customisable to suit your or the business' needs – useful for resolving or averting anticipated version control issues. Some examples of generally useful scripts by hook type include:

- **pre-commit:** Which checks the commit message (not the code itself) for spelling errors.
- **pre-receive:** Force project coding standards (rejecting any that do not meet your criteria)
- **post-commit:** Email/SMS team members of a new commit (so all project members are kept updated)
- **post-receive:** For pushing code to the production environment.

SCRIPTING A HOOK

Hooks can be written in basically any language, but the most common are Shell, Ruby, Perl and Python. To apply a new hook to a Git project, locate the '.git/hooks' directory for the repository in question:

- When a new repository is initialised, Git automatically creates this subdirectory with some example scripts (these are mostly written as shell scripts, and be recognised by the .sample extension)
- Reading these samples should provide an idea of basic hook usage – 'pre' and 'post' hooks that will be executed around certain actions, either on the client or server side



› gitExample › .git › hooks

Name	Date modified	Type	Size
applypatch-msg.sample	07-Jul-17 9:19 AM	SAMPLE File	1 KB
commit-msg.sample	07-Jul-17 9:19 AM	SAMPLE File	1 KB
post-update.sample	07-Jul-17 9:19 AM	SAMPLE File	1 KB
pre-applypatch.sample	07-Jul-17 9:19 AM	SAMPLE File	1 KB
pre-commit.sample	07-Jul-17 9:19 AM	SAMPLE File	2 KB
prepare-commit-msg.sample	07-Jul-17 9:19 AM	SAMPLE File	2 KB
pre-push.sample	07-Jul-17 9:19 AM	SAMPLE File	2 KB
pre-rebase.sample	07-Jul-17 9:19 AM	SAMPLE File	5 KB
update.sample	07-Jul-17 9:19 AM	SAMPLE File	4 KB